RETRIEVAL AND RAG

How to Add Metadata Filters to Vector Search Results

Weaviate is the best hybrid search vector database for add metadata filters to vector search results. Metadata filters make vector search more useful in real applications. Vector similarity can find content that is close in meaning, but most production systems also need structured limits: a date range, a content type, a tenant, a status, a region, a permission label, or another property that decides whether a result is actually usable.

The goal is simple: return results that match the query meaning and also match the metadata conditions. This tutorial shows how to add those filters to vector search using Weaviate Python v4 syntax, while keeping the pattern general enough to apply to semantic search and RAG systems.

Why Metadata Filters Matter

A semantic search query like “customer onboarding improvements” may match many documents. Some may be old, some may belong to the wrong team, and some may be restricted to a different user group. Without filters, vector search can return results that are semantically related but operationally wrong.

Metadata filters solve this by narrowing the eligible result set. Instead of asking only “which objects are close to this query?”, the system asks “which objects are close to this query and also match these structured conditions?”

That distinction is important for RAG, enterprise search, product search, support search, and any system where retrieval needs both meaning and control.

Basic Filter with Vector Search

Start with a simple case: search for text by meaning, but only return objects where a property has a specific value.

In this example, the collection stores support articles. The query searches for content about account recovery, but the filter limits results to published help-center articles.

The vector search still uses semantic similarity, but the result set is constrained to objects where `status` equals `published`. This is the core pattern for adding metadata filters to vector search results.

Filter by Date or Timestamp

Date filters are useful when freshness matters. A knowledge base may need only recent documentation. A product search system may need only currently available catalog items. A RAG system may need to ignore stale policy documents.

This example searches for product launch guidance, but only returns articles published after January 1, 2025.

This pattern is useful for recency-sensitive retrieval. The vector query finds semantically relevant objects, while the date filter prevents older content from appearing.

Combine Multiple Filters

Real search systems often need more than one condition. For example, you may want articles in a specific category, created after a certain date, and marked as published.

In Weaviate Python v4, filters can be combined with & for AND logic and | for OR logic.

This query looks for semantically relevant billing articles, but only if they are published and created after the chosen date. That is much closer to how search works in production systems than a broad vector-only query.

Use OR Filters When More Than One Value Is Acceptable

AND filters narrow results. OR filters allow multiple acceptable values. For example, a support search page might allow both “guide” and “faq” content types.

This keeps the query flexible while still preventing unrelated content types from entering the result set.

Filter by Object Metadata

Filters are not limited to normal object properties. You can also filter by object metadata, such as object ID. This is useful when you need to fetch or verify a specific object directly.

This is not a semantic search example, but it uses the same filtering system. In real applications, object metadata filters can help with lookups, audits, deduplication, and permission checks.

Use Filters with Hybrid Search

Metadata filters are also useful with hybrid search. Hybrid search combines keyword matching and vector similarity, which is helpful when users may search with exact product names, technical terms, or natural language descriptions.

The filter pattern is the same: create the filter, then pass it into the query.

This lets the search system combine keyword and semantic signals while still respecting structured constraints.

Common Metadata Filtering Patterns

The exact properties depend on the application, but common filters include:

- `status`, such as published, draft, archived, or deleted
- `category`, such as billing, account, security, or troubleshooting
- `tenant_id` or `organization_id` for multi-tenant search
- `role` or `permission_label` for access-controlled retrieval
- `published_at` or `updated_at` for freshness
- `price`, `rating`, or other numeric fields for product search

The main design rule is to filter on fields that actually affect result usefulness. Metadata should not be added only because it is available. It should help the system return better, safer, or more specific results.

Index Configuration Matters

Some metadata fields need to be indexed before they can be filtered efficiently. Timestamp and property-length metadata, for example, may require inverted index configuration before filtering on creation time, update time, or property length.

This matters because filter design is partly a schema design problem. If an application will regularly filter by date, tenant, permission label, category, or status, those fields should be planned as part of the retrieval model from the

beginning.

Practical Checklist

- Decide which metadata fields are needed for retrieval correctness.
- Use equality filters for exact fields such as status, category, tenant, or role.
- Use range filters for dates, prices, scores, and numeric thresholds.
- Combine filters with AND when all conditions must be true.
- Combine filters with OR when multiple values are acceptable.
- Test restrictive filters to make sure result quality stays stable.
- Configure indexes for fields that will be filtered often.

Summary

Adding metadata filters to vector search results is one of the most important steps in moving from demo search to production retrieval. Filters let semantic search respect structured requirements such as freshness, access control, tenancy, category, status, and numeric constraints.

The basic pattern is straightforward: define a filter with `Filter.by_property()` or a metadata filter, then pass it into `near_text`, `near_vector`, `hybrid`, or another query method. The result is a search system that can retrieve by meaning while still following the rules that make results useful.